Volume 4, Issue 12, December 2014 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Lean Software Development – Survey on Benefits and Challenges in Agile and Lean Usage in Small and medium Software Firms in Bangalore

[1]Piyush Kumar Pareek, [2]Dr. A. N. Nandakumar
[1]Assistant Professor, Department of CSE, Kammavari Institute of Technology, Bengaluru & Research Scholar, Jain University, Bengaluru, India
[2]Principal, R.L.Jalappa institute of Technology & Research Guide, Jain University, Bengaluru, India

Abstract*— Lean Software Development tells us we need to Limit work to capacity to remove delays and lower interruptions Attend to the timing of the workflow with methods such as acceptance test-driven development, test-driven development and continuous integration Use time-to-market (which Lean calls "cycle time") as the ultimate measure of progress . The objective of this paper was to identify most experienced benefits and challenges of using agile software development and lean software development.

Keywords*— agile software development and lean software development

I. INTRODUCTION

Lean software development (LSD) is a translation of lean manufacturing and lean IT principles and practices to the software development domain. Adapted from the Toyota Production System, a pro-lean subculture is emerging from within the agile community. Lean development can be summarized by seven principles, very close in concept to lean manufacturing principles:

1. Eliminate waste
2. Amplify learning
3. Decide as late as possible
4. Deliver as fast as possible
5. Empower the team
6. Build integrity in
7. See the whole

Eliminate waste

Everything not adding value to the customer is considered to be waste (*muda*). This includes:

- unnecessary code and functionality
- delay in the software development process
- unclear requirements
- insufficient testing, leading to avoidable process repetition
- bureaucracy
- slow internal communication

In order to be able to eliminate waste, one should be able to recognize it. If some activity could be bypassed or the result could be achieved without it, it is waste. Partially done coding eventually abandoned during the development process is waste. Extra processes and features not often used by customers are waste. Waiting for other activities, teams, processes is waste. Defects and lower quality are waste. Managerial overhead not producing real value is waste. A value stream mapping technique is used to distinguish and recognize waste. The second step is to point out sources of waste and eliminate them. The same should be done iteratively until even essential-seeming processes and procedures are liquidated.

Amplify learning

Software development is a continuous learning process with the additional challenge of development teams and end product sizes. The best approach for improving a software development environment is to amplify learning. The accumulation of defects should be prevented by running tests as soon as the code is written. Instead of adding more documentation or detailed planning, different ideas could be tried by writing code and building. The process of user requirements gathering could be simplified by presenting screens to the end-users and getting their input.

The learning process is sped up by usage of short iteration cycles – each one coupled with refactoring and integration testing. Increasing feedback via short feedback sessions with customers helps when determining the current phase of development and adjusting efforts for future improvements. During those short sessions both customer

representatives and the development team learn more about the domain problem and figure out possible solutions for further development. Thus the customers better understand their needs, based on the existing result of development efforts, and the developers learn how to better satisfy those needs. Another idea in the communication and learning process with a customer is set-based development – this concentrates on communicating the constraints of the future solution and not the possible solutions, thus promoting the birth of the solution via dialogue with the customer.

Decide as late as possible

As software development is always associated with some uncertainty, better results should be achieved with an options-based approach, delaying decisions as much as possible until they can be made based on facts and not on uncertain assumptions and predictions. The more complex a system is, the more capacity for change should be built into it, thus enabling the delay of important and crucial commitments. The iterative approach promotes this principle – the ability to adapt to changes and correct mistakes, which might be very costly if discovered after the release of the system.

An agile software development approach can move the building of options earlier for customers, thus delaying certain crucial decisions until customers have realized their needs better. This also allows later adaptation to changes and the prevention of costly earlier technology-bounded decisions. This does not mean that no planning should be involved – on the contrary, planning activities should be concentrated on the different options and adapting to the current situation, as well as clarifying confusing situations by establishing patterns for rapid action. Evaluating different options is effective as soon as it is realized that they are not free, but provide the needed flexibility for late decision making.

Deliver as fast as possible

In the era of rapid technology evolution, it is not the biggest that survives, but the fastest. The sooner the end product is delivered without major defects; the sooner feedback can be received, and incorporated into the next iteration. The shorter the iterations, the better the learning and communication within the team. With speed, decisions can be delayed. Speed assures the fulfilling of the customer's present needs and not what they required yesterday. This gives them the opportunity to delay making up their minds about what they really require until they gain better knowledge. Customers value rapid delivery of a quality product.

The just-in-time production ideology could be applied to software development, recognizing its specific requirements and environment. This is achieved by presenting the needed result and letting the team organize itself and divide the tasks for accomplishing the needed result for a specific iteration. At the beginning, the customer provides the needed input. This could be simply presented in small cards or stories – the developers estimate the time needed for the implementation of each card. Thus the work organization changes into self-pulling system – each morning during astand-up meeting, each member of the team reviews what has been done yesterday, what is to be done today and tomorrow, and prompts for any inputs needed from colleagues or the customer. This requires transparency of the process, which is also beneficial for team communication. Another key idea in Toyota's Product Development System is set-based design. If a new brake system is needed for a car, for example, three teams may design solutions to the same problem. Each team learns about the problem space and designs a potential solution. As a solution is deemed unreasonable, it is cut. At the end of a period, the surviving designs are compared and one is chosen, perhaps with some modifications based on learning from the others - a great example of deferring commitment until the last possible moment. Software decisions could also benefit from this practice to minimize the risk brought on by big up-front design.

Empower the team

There has been a traditional belief in most businesses about the decision-making in the organization – the managers tell the workers how to do their own job. In a Work-Out technique, the roles are turned – the managers are taught how to listen to the developers, so they can explain better what actions might be taken, as well as provide suggestions for improvements. The lean approach favors the aphorism "find good people and let them do their own job," encouraging progress, catching errors, and removing impediments, but not micro-managing.

Another mistaken belief has been the consideration of people as resources. People might be resources from the point of view of a statistical data sheet, but in software development, as well as any organizational business, people do need something more than just the list of tasks and the assurance that they will not be disturbed during the completion of the tasks. People need motivation and a higher purpose to work for – purpose within the reachable reality, with the assurance that the team might choose its own commitments. The developers should be given access to the customer; the team leader should provide support and help in difficult situations, as well as ensure that skepticism does not ruin the team's spirit.

Build integrity in

The customer needs to have an overall experience of the System – this is the so-called perceived integrity: how it is being advertised, delivered, deployed, accessed, how intuitive its use is, price and how well it solves problems.

Conceptual integrity means that the system's separate components work well together as a whole with balance between flexibility, maintainability, efficiency, and responsiveness. This could be achieved by understanding the problem domain and solving it at the same time, not sequentially. The needed information is received in small batch pieces – not in one vast chunk with preferable face-to-face communication and not any written documentation. The information flow should be constant in both directions – from customer to developers and back, thus avoiding the large stressful amount of information after long development in isolation.

One of the healthy ways towards integral architecture is refactoring. As more features are added to the original code base, the harder it becomes to add further improvements. Refactoring is about keeping simplicity, clarity, minimum amount of features in the code. Repetitions in the code are signs for bad code designs and should be avoided. The complete and automated building process should be accompanied by a complete and automated suite of developer and customer tests, having the same versioning, synchronization and semantics as the current state of the System. At the end the integrity should be verified with thorough testing, thus ensuring the System does what the customer expects it to. Automated tests are also considered part of the production process, and therefore if they do not add value they should be considered waste. Automated testing should not be a goal, but rather a means to an end, specifically the reduction of defects.

See the whole

Software systems nowadays are not simply the sum of their parts, but also the product of their interactions. Defects in software tend to accumulate during the development process – by decomposing the big tasks into smaller tasks, and by standardizing different stages of development, the root causes of defects should be found and eliminated. The larger the system, the more organizations that are involved in its development and the more parts are developed by different teams, the greater the importance of having well defined relationships between different vendors, in order to produce a system with smoothly interacting components. During a longer period of development, a stronger subcontractor network is far more beneficial than short-term profit optimizing, which does not enable win-win relationships.

Lean thinking has to be understood well by all members of a project, before implementing in a concrete, real-life situation. "Think big, act small, fail fast; learn rapidly" – these slogans summarize the importance of understanding the field and the suitability of implementing lean principles along the whole software development process. Only when all of the lean principles are implemented together, combined with strong "common sense" with respect to the working environment, is there a basis for success in software development.

Lean software Practice

Lean software development practices, or what the Poppendiecks call "tools" are expressed slightly differently from their equivalents in agile software development, but there are parallels. Examples of such practices include:

- Seeing waste
- Value stream mapping
- Set-based development
- Pull systems
- Queuing theory
- Motivation
- Measurements

II. LITERATURE SURVEY

Lean software development is the application of the principles of the Toyota product development system to software development. Toyota has been extremely successful developing complex new vehicles, which include a vast amount of embedded software, in a very short time and always on time. This tutorial examines the underlying engineering principles Toyota uses to develop vehicles and shows how they can be applied to software development. When correctly applied, lean software development results in high quality software that is developed quickly and at the lowest possible cost. Moreover, the success of many of the practices of Agile Software Development can be explained by understanding the principles of Lean Software development [1].

Lean principles, originating from Japanese automotive industry, are anticipated to be useful to improve software development processes. Albeit its popularity there is still no generally accepted, clear and detailed definition of what lean software development actually means. This makes it difficult to perform research on the effects of lean software development and determine its usefulness in various contexts. To fill in that research gap this paper analyzes the state of the art based on twenty key Lean concepts derived from nine seminal sources identified in a systematic literature review. The original explanations of the key concepts have been elaborated further and synthesized into a framework for lean software development consisting of a set of goals, recommended activities and practices. The detailed results for the key concept Value are reported [2].

The failure mode and effects analysis (FMEA) technology is one of the basic methods in the process of designing and analyzing the reliability. In view of the poor development for FMEA analysis tools of the army equipment, a new equipment FMEA software system is designed according to the special function requirements of equipment FMEA software on the basis of the FMEA software development technology at home and abroad [3].

Value stream mapping is a kind of technique that helps to understand and streamline production processes. With a case study of a metal machinery factory, the production process path is visualized by mapping the current state value stream. After tracking the production process of a medium-sized order, problems affecting the delivery time are identified and its causes analyzed. A future state value stream map is created and an optimization scheme is suggested, with which the production cycle of the order is expected to be shortened from 21 days to 9 days, representing a 57% reduction [4].

By using the waste-value concept and related tools an organization can continuously measure and improve its processes. The main analysis tool is the value map stream, which maps the value adding activities of the SDP, and the seven wastes of lean software development, a set of waste activities that are commonly seen in. The lean value map is

used to give an overall view of the process, and the seven wastes serves as a guide to focus the effort of finding waste. It has been shown, as is also discussed in the introduction, that agile approaches seldom decrease performance over a traditional manufacturing-inspired approach. Therefore, by using light-weight agile methodologies as a good practice comparison, arguments can be presented how waste in the process can be reduced. A number of sources contributed to my interpretation of agile and lean, and their unification in the approach called lean software development [5].

III. RESEARCH METHODS USED DURING SURVEY

Extensive Exploratory web survey study including almost 25 questions conducted among Small and medium Software Firms in Bangalore, 500 software engineers from almost 50 Small and medium Software Firms participated in this. Data Analysis was done using SPSS Version 21.0

IV. DATA ANALYSIS

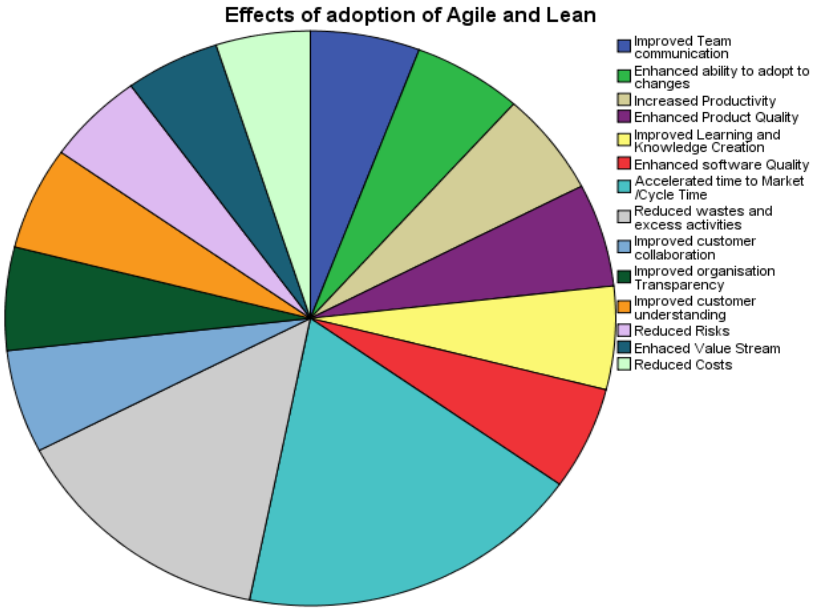

Figure 1: Effects of adoption of Agile and Lean

Table 1 : Effects of adoption of Agile and Lean

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Improved Team communication	29	5.8	5.8	5.8
	Enhanced ability to adopt to changes	29	5.8	5.8	11.6
	Increased Productivity	29	5.8	5.8	17.4
	Enhanced Product Quality	29	5.8	5.8	23.2
	Improved Learning and Knowledge Creation	29	5.8	5.8	29.0
	Enhanced software Quality	29	5.8	5.8	34.8
	Accelerated time to Market /Cycle Time	92	18.4	18.4	53.2
	Reduced wastes and excess activities	71	14.2	14.2	67.4
	Improved customer collaboration	29	5.8	5.8	73.2
	Improved organisation Transparency	29	5.8	5.8	79.0
	Improved customer understanding	29	5.8	5.8	84.8
	Reduced Risks	26	5.2	5.2	90.0
	Enhaced Value Stream	25	5.0	5.0	95.0
	Reduced Costs	25	5.0	5.0	100.0
	Total	500	100.0	100.0	

As it can be observed from Figure 1 and Table 1 that Effects of adopting Agile and Lead to Accelerated delivery time ,Reduction of cycle time ,Reduction in Wastes and additional activities.

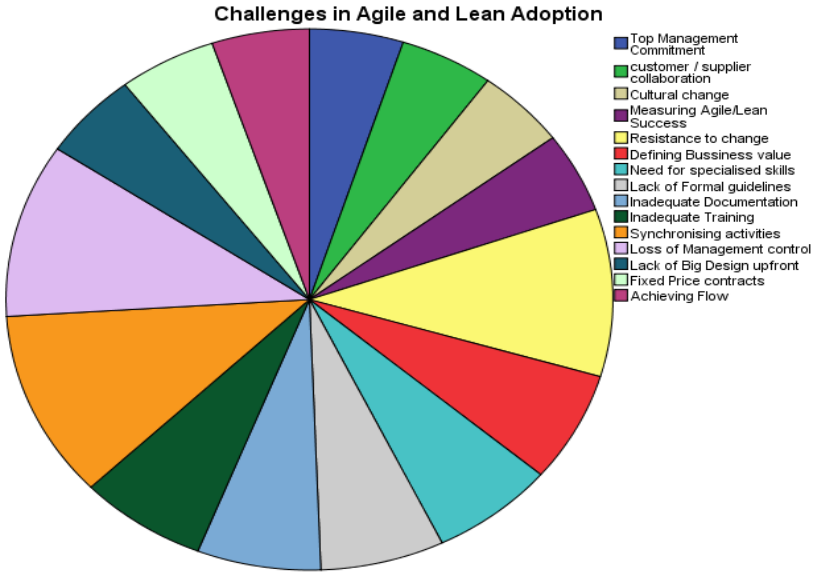

Figure 2: Challenges in Implementing Lean and Agile

Table 2 : Challenges in Agile and Lean Adoption

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Top Management Commitment	25	5.0	5.0	5.0
	customer / supplier collaboration	25	5.0	5.0	10.0
	Cultural change	24	4.8	4.8	14.8
	Measuring Agile/Lean Success	24	4.8	4.8	19.6
	Resistance to change	50	10.0	10.0	29.6
	Defining Bussiness value	33	6.6	6.6	36.2
	Need for specialised skills	33	6.6	6.6	42.8
	Lack of Formal guidelines	33	6.6	6.6	49.4
	Inadequate Documentation	33	6.6	6.6	56.0
	Inadequate Training	34	6.8	6.8	62.8
	Synchronising activities	56	11.2	11.2	74.0
	Loss of Management control	52	10.4	10.4	84.4
	Lack of Big Design upfront	26	5.2	5.2	89.6
	Fixed Price contracts	26	5.2	5.2	94.8
	Achieving Flow	26	5.2	5.2	100.0
	Total	500	100.0	100.0	

As it can be observed from Figure 2 and Table 2 that synchronising all activities,resistance to change and adopt new processes and loss of management control were main challenges that these firms encountered.

Table 3 : Limitations in Agile and Lean Production

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Limited support for developing Large nd Complex software	167	33.4	33.4	33.4
	Lmited support for developing involving Large Teams	41	8.2	8.2	41.6

	Limited support for subcontracting	42	8.4	8.4	50.0
	Limited support for developing Legacy systems	42	8.4	8.4	58.4
	Limited support for distributed developing environments	83	16.6	16.6	75.0
	Limited support for building reusable artifacts	42	8.4	8.4	83.4
	Limited support for developing critical systems	83	16.6	16.6	100.0
	Total	500	100.0	100.0	

From Table 3 , it can be observed that its a limitation of using Lean that large complex systems very difficult to adopt it .

V. RESULTS

1. About 18 % of the respondents accepted that due to adoption of Lean they reduced cycle time and reduced market delivery time
2. About 14 % observed reduction of wastes after adoption of Lean
3. About 11 % of respondents say synchronising of activities was a challenge in implementing Lean
4. About 20 % of respondents agree on Loss of management control and resistance to change as major challenge in adoption of Lean
5. About 33 % of respondents say limitation of using Agile is it provides limited support for developing large and complex softwares.

VI. CONCLUSION

Lean adoption leads to reduction of wastes , In today's competitive environment usage of Lean can lead to increase in productivity and improving quality, Lean can further be adopted in website developing companies to reduce non value added activities in their development as most of the small and medium level enterprises have the website development business commonly adopted .

REFERENCES

[1] Lean Software Development by Poppendieck, M. et al, Software Engineering - Companion, 2007. ICSE 2007 Companion. 29th International Conference on Digital Object Identifier: 10.1109/ICSECOMPANION.2007.46 Publication Year: 2007, Page(s): 165 - 166, IEEE CONFERENCE PUBLICATIONS.

[2] Synthesizing a Comprehensive Framework for Lean Software Development Jonsson, H. Larsson, S. Punnekkat et al, Software Engineering and Advanced Applications (SEAA), 2013 39th EUROMICRO Conference on Digital Object Identifier: 10.1109/SEAA.2013.64 Publication Year: 2013 , Page(s): 1 – 8, IEEE CONFERENCE PUBLICATIONS

[3] Design and development of the software system for equipment FMEA based on .NET ,.Lv Yan-mei ,Wang Ge-fang ,Wang Haiti , Wang Jialing ,Wang Wei-hue Mechatronics and Automation (ICMA), 2012 International Conference on Digital Object Identifier: 10.1109/ICMA.2012.6285745 , Publication Year: 2012 , Page(s): 2531 – 2535, IEEE CONFERENCE PUBLICATIONS.

[4] Application research of shortening delivery time through value stream mapping analysis Gouging Pan , Ding-song Fang , Mei-Xian Jiang ,Industrial Engineering and Engineering Management (IE&EM), 2010 IEEE 17Th International Conference on Digital Object Identifier: 10.1109/ICIEEM.2010.5646515 ,Publication Year: 2010 , Page(s): 733 - 736 , IEEE CONFERENCE PUBLICATIONS.

[5] Lean Software Development: Poppendieck, M., & Poppendieck, T. (2009), Crawfordsville: Addison-Wesley Professional.

